Chap. 4

ARM Organization and Implementation

4.1 3-stage pipeline ARM organization

- The 3-stage pipeline
  - ARM processors up to the ARM7 employ a simple 3-stage pipeline with the following pipeline stages: (Fig. 4.1)
    - *Fetch*: The instruction is fetched from memory and placed in the instruction pipeline
    - *Decode*: The instruction is decoded and the datapath control signals prepared for the next cycle
    - *Execute*: The register bank is read, an operand shifted, the ALU result generated and written back into a destination register
  - An individual instruction takes three clock cycles to complete, so it has a three-cycle latency, but the throughput is one instruction per cycle (Fig. 4.2)

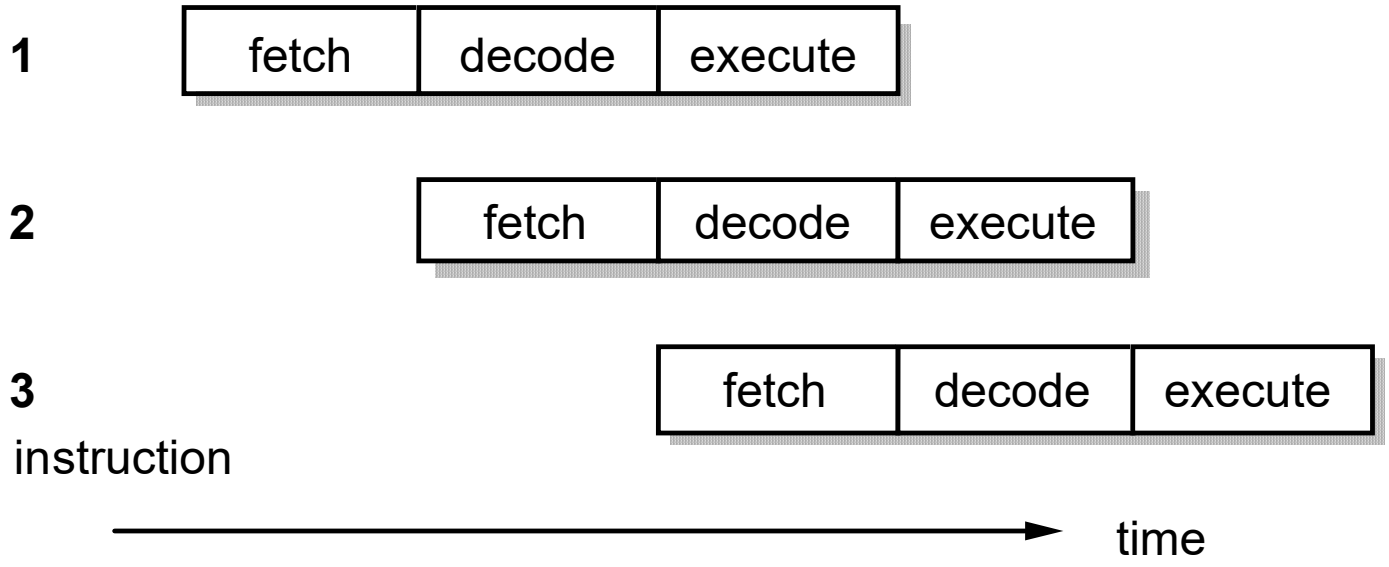
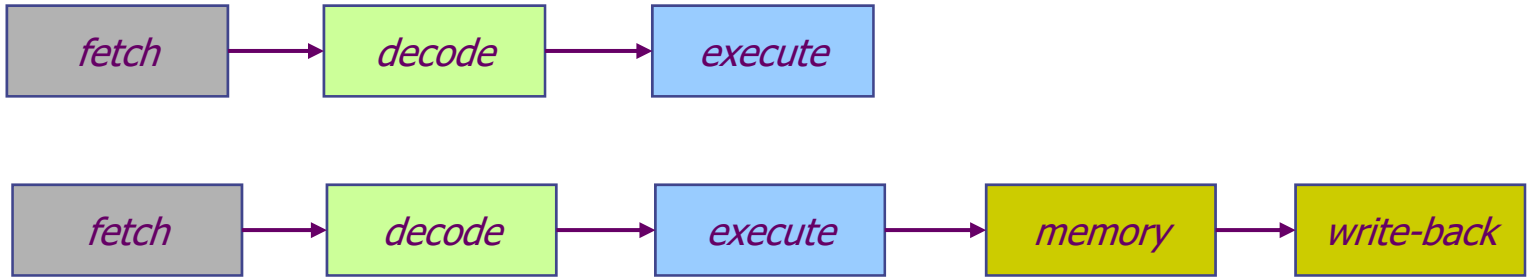


Fig. 4.2 ARM single-cycle instruction pipeline operation

Pipeline

- 3 stages (ARM7) and 5 stages (ARM9TDMI)



PC
Load an instruction from memory

PC-4

PC-8

access memory if needed

write result to register

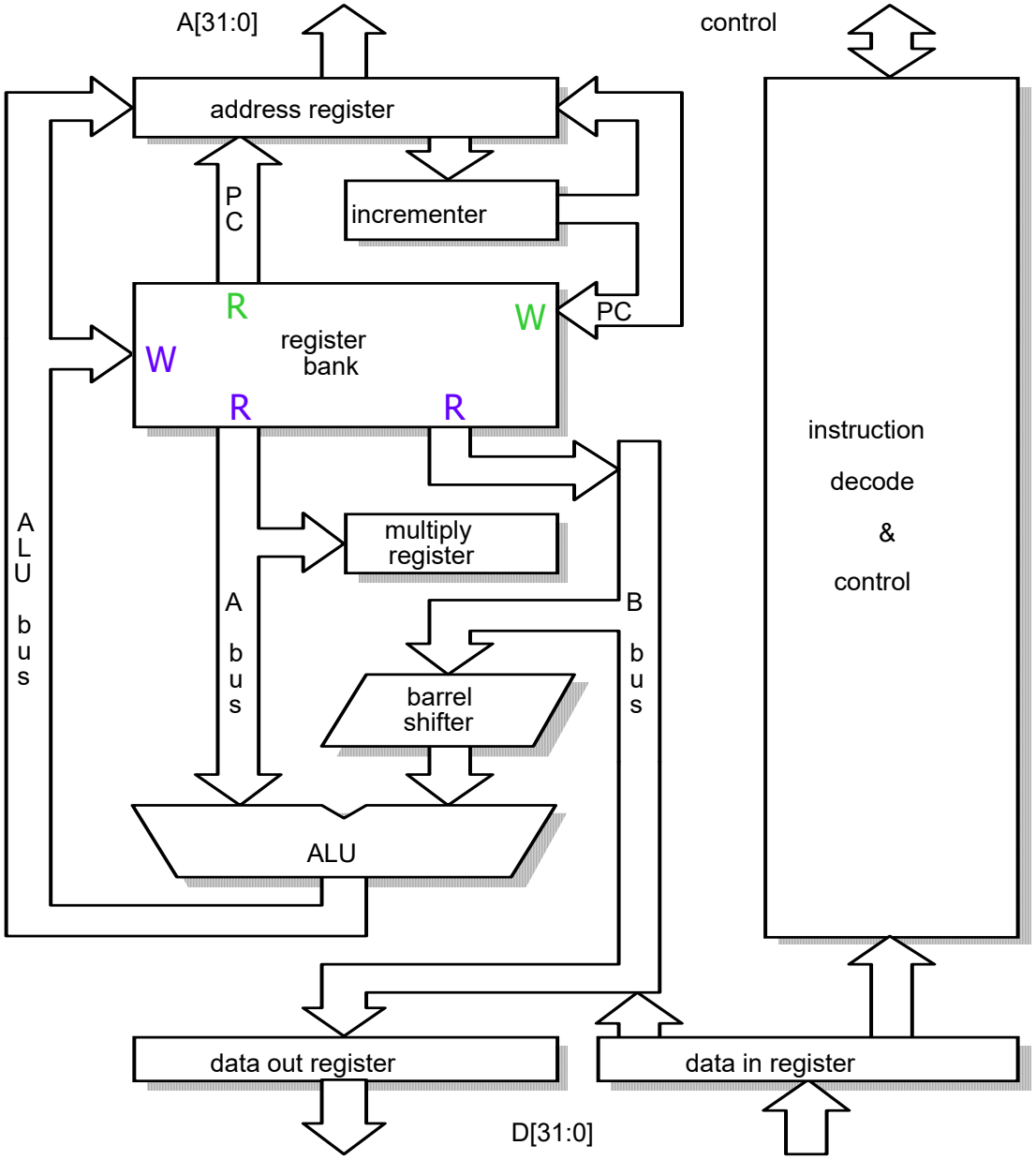


Fig. 4.1 ARM organization

4.1 3-stage pipeline ARM organization

- The 3-stage pipeline (Cont.)
  - When a multi-cycle instruction is executed the flow is less regular

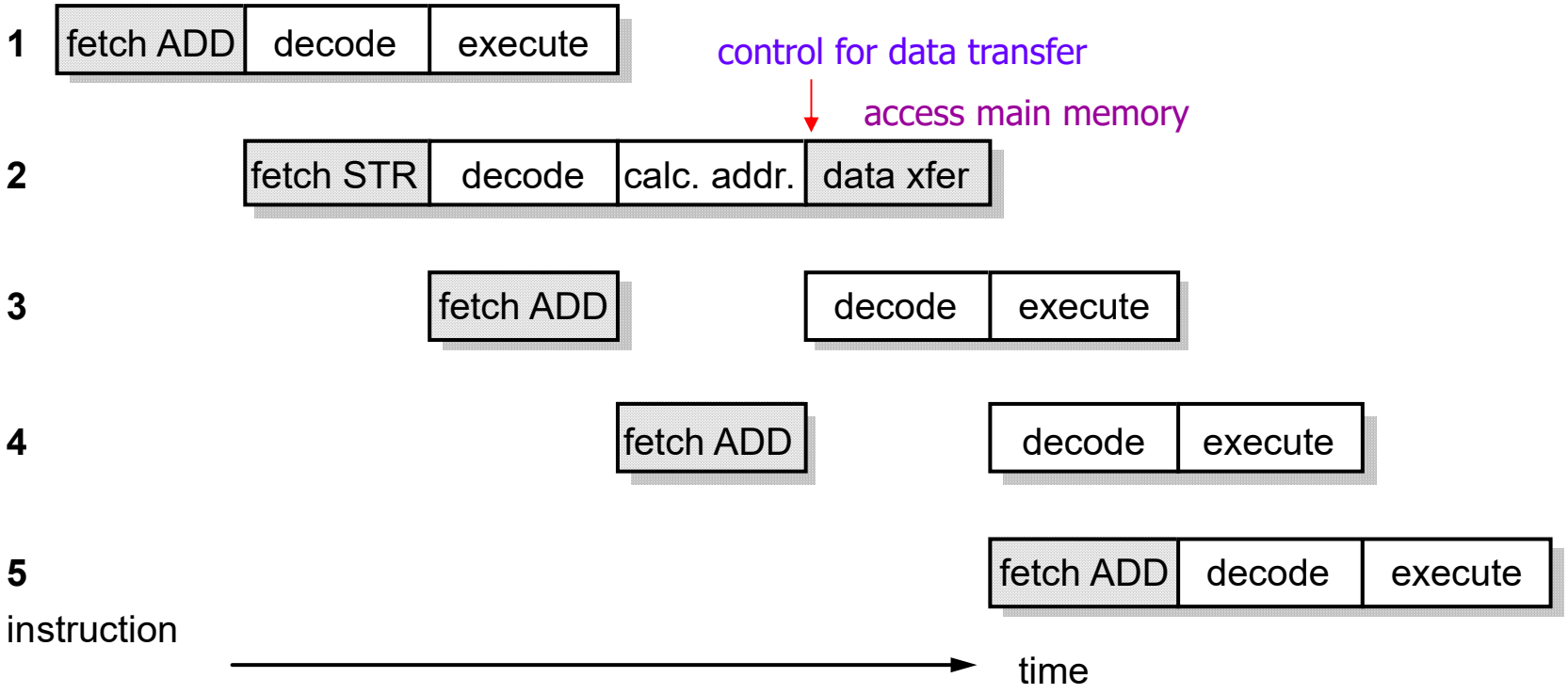


Fig. 4.3 ARM multi-cycle instruction pipeline operation

4.1 3-stage pipeline ARM organization

- The 3-stage pipeline (Cont.)
  - All instructions occupy the datapath for one or more adjacent cycles
  - For each cycle that an instruction occupies the datapath, it occupies the decode logic in the immediately preceding cycle
  - During the first datapath cycle each instruction issues a fetch for the next instruction but one
  - Branch instructions flush and refill the instruction pipeline

- Due to the pipelined execution model used on the ARM is that the program counter, which is visible to the user as r15, must run ahead of the current instruction.
  - PC must point eight bytes ahead of the current executed instruction.
  - In Chapter5:
    - Specifying the PC in an STM should be avoided

5.2 Exceptions

- Exception return
  - Any modified user registers must be restored from the handler's stack
  - The CPSR must be restored from the appropriate SPSR
  - The PC must be changed back to the relevant instruction address in the user instruction stream
  - The last two steps cannot be carried out independently.
  - The case where the return address is in r14 (S: destination is pc):
    - To return from a SWI or undefined instruction trap use

      MOVS pc, r14
    - To return from an IRQ, FIQ or prefetch abort use

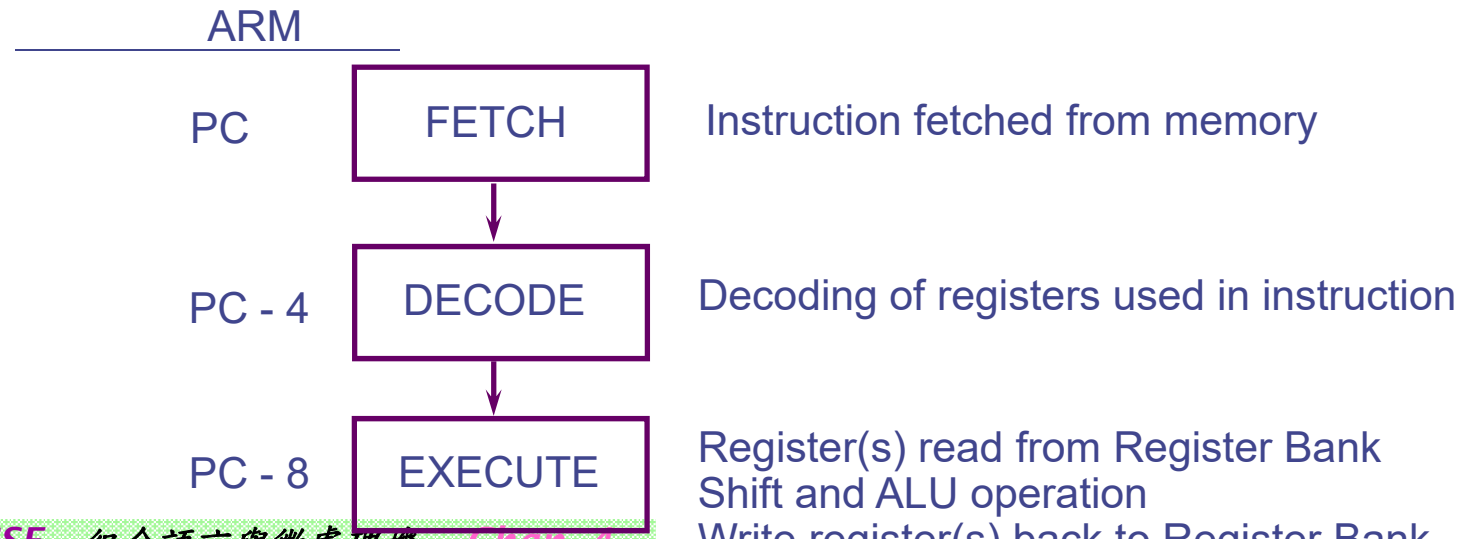
      SUBS pc, r14, #4
    - To return from a data abort to retry the data access use

      SUBS pc, r14, #8
  - The return address out onto a stack:
    - LDMFD r13!, {r0 – r3, pc}^ ; restore and return

CPSR is also restore

The Instruction Pipeline

Exception	*Address*	*Use*
Reset	-	lr is not defined
Data abort	lr-8	Points to the instructions that caused data abort
FIQ	lr-4	Return to next instruction
IRQ	lr-4	Return to next instruction
Prefetch Abort	lr-4	Points to the instruction that cause prefetch aboint
SWI	lr	Return to next instruction after SWI
Undefined Instruction	lr	Return to next instruction after undefined instruction



4.2 5-stage pipeline ARM organization

- 3-stage (cost-effective) ⇒ 5-stage (higher performance)
  - The time, T_{prog}, required to execute a given program is given by:

$$T_{prog} = N_{inst} \times CPI / f_{clk}$$

    - N_{inst} : number of ARM instructions executed in the course of the program
    - *CPI* : the average number of clock cycles per instruction
    - f_{clk} : the processor's clock frequency

  - There are two ways to increase performance:
    - Increase the clock rate, f_{clk}
      - Logic in each pipeline stage to be simplified ⇒ the number of pipeline stage increased
    - Reduce the average number of clock cycles per instruction, *CPI*
      - Instructions which occupy more than one pipeline slot in a 3-stage pipeline ARM are re-implemented to occupy fewer slots
      - Pipeline stalls caused by dependencies between instruction are reduced

4.2 5-stage pipeline ARM organization

- Memory bottleneck
  - Any stored-program computer with a single instruction and data memory will have its performance limited by the available memory bandwidth
  - To get a significantly better CPI the memory system must deliver more than one value in each clock cycle either by delivering more than 32 bits per cycle from a single memory or by having separate memories for instruction and data accesses
  - Higher performance ARM cores employ a 5-stage pipeline and have separate instruction and data memories

4.2 5-stage pipeline ARM organization

- The 5-stage pipeline
  - ***Fetch***
    - the instruction is fetched from memory and placed in the instruction pipeline
  - ***Decode***
    - the instruction is decoded and register operands read from the register file
  - ***Execute***
    - an operand is shifted and the ALU result generated
  - ***Buffer/data***
    - data memory is accessed if required
    - otherwise the ALU result is simply buffered for one clock cycle
  - ***Write-back***
    - the results generated by the instruction are written back to the register file

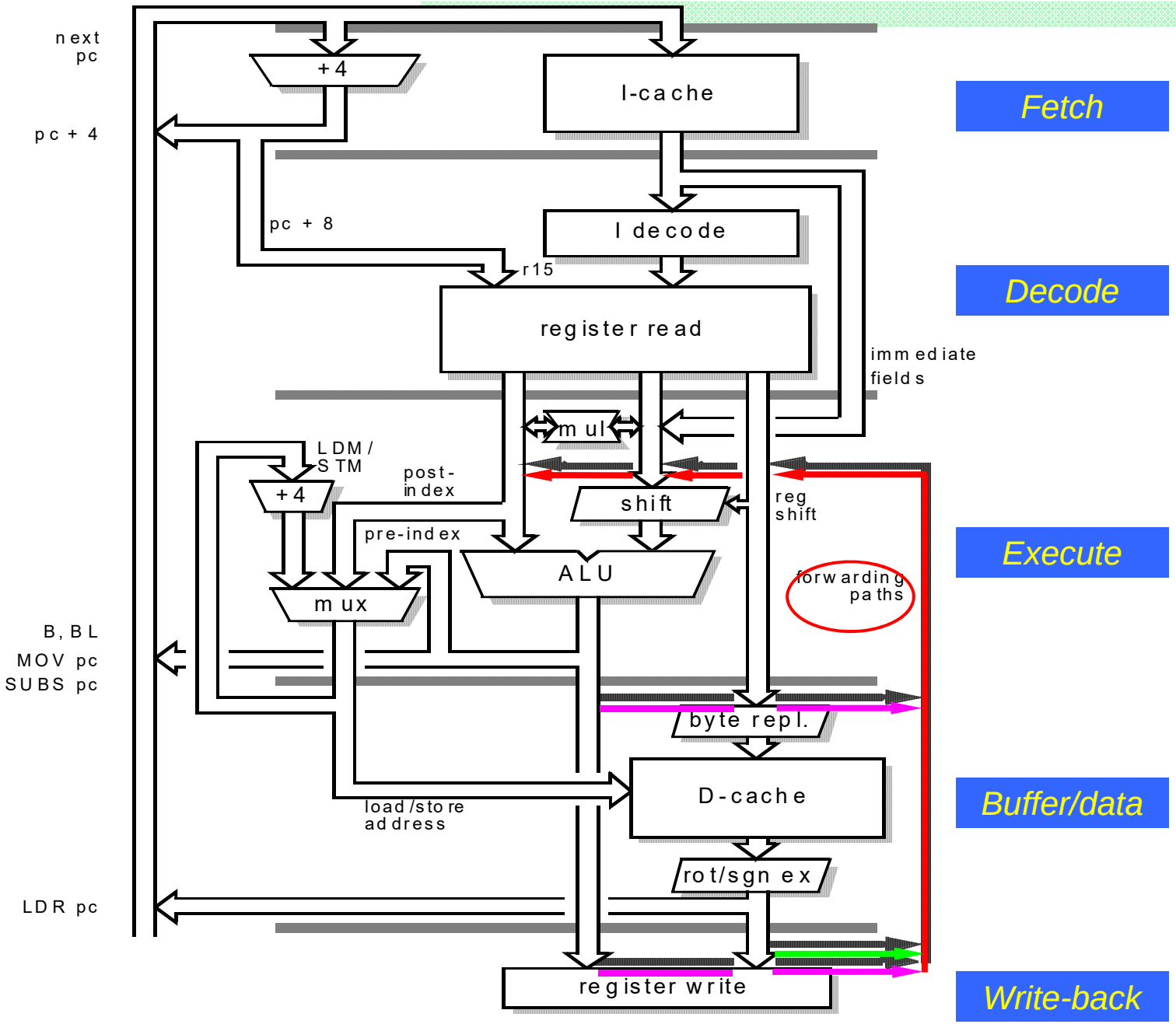


Fig. 4.4 ARM9TDMI 5-stage pipeline organization

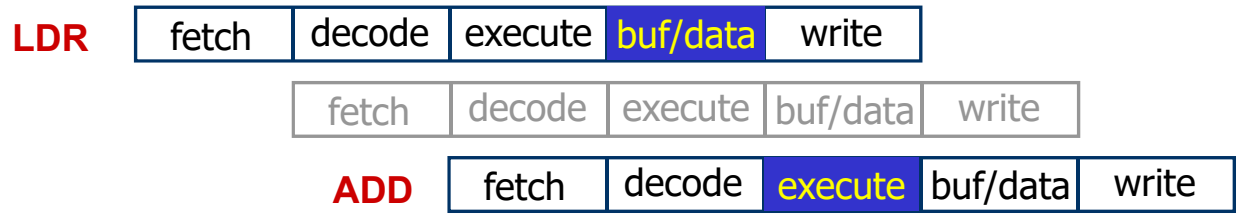
4.2 5-stage pipeline ARM organization

Data forwarding

- Because instruction execution is spread across three pipeline stages, the only way to resolve data dependencies without stalling the pipeline is to introduce forwarding paths
- Forwarding paths allow results to be passed between stages as soon as they are available
- Consider the following code sequence:

```
LDR     rN,  [ . . ]      ; load rN from somewhere
ADD     r2, r1, rN        ; and use it immediately
```

- The only way to avoid this stall is to encourage the compiler not to put a dependent instruction immediately after a load instruction

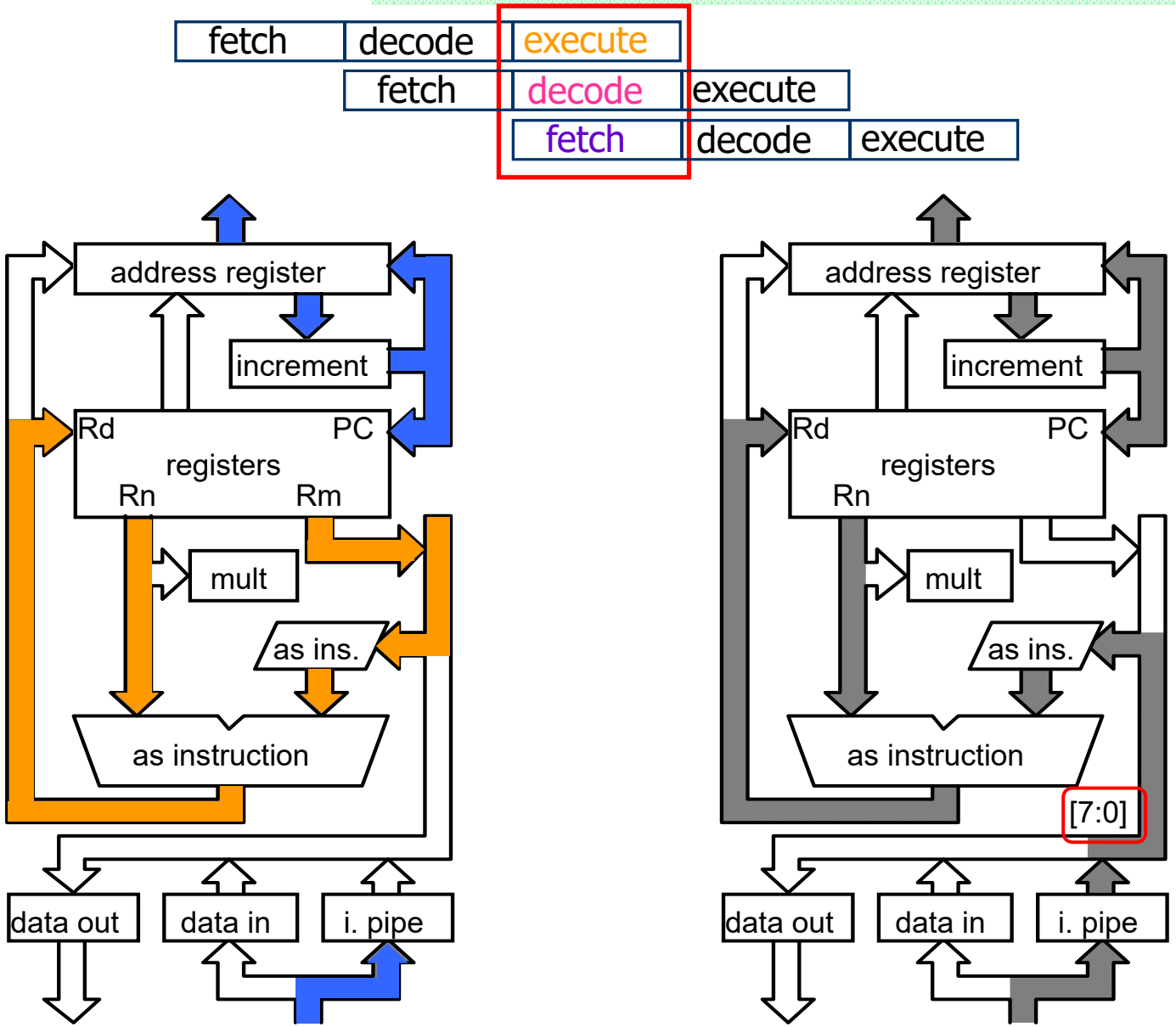


4.3 ARM instruction execution

- Data path organization: **Fig. 4.1** (3-stage pipeline)
- **Data processing instructions**
  - A data processing instruction requires two operands, one of which is always a register and the other is either a second register or an immediate value
  - The second operand is passed through the barrel shifter where it is subject to a general shift operation

```
ADD     r3,  r2,  r1,  LSL #3       ; r3 := r2 + 8 × r1
```

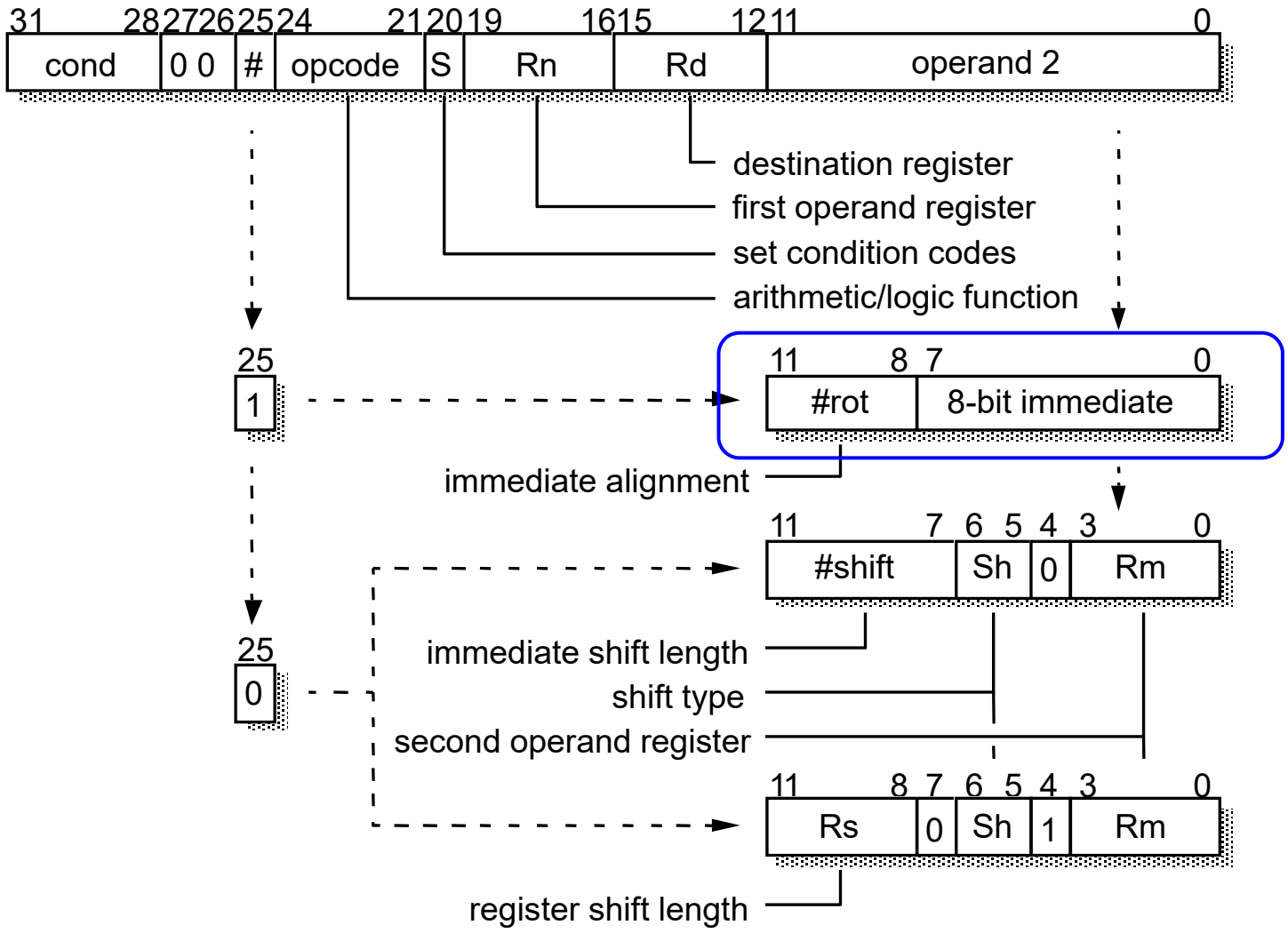
  - The result from the ALU is written back into the destination register (the condition code register may be updated)
  - All these operations take place in a single clock cycle as shown in **Fig. 4.5**
  - Only the bottom eight bits (bits [7:0]) of the instruction are used in the immediate value



(a) r egister - r egister operations

(b) r egister - immediate operations

Fig. 4.5 Data processing instruction datapath activity



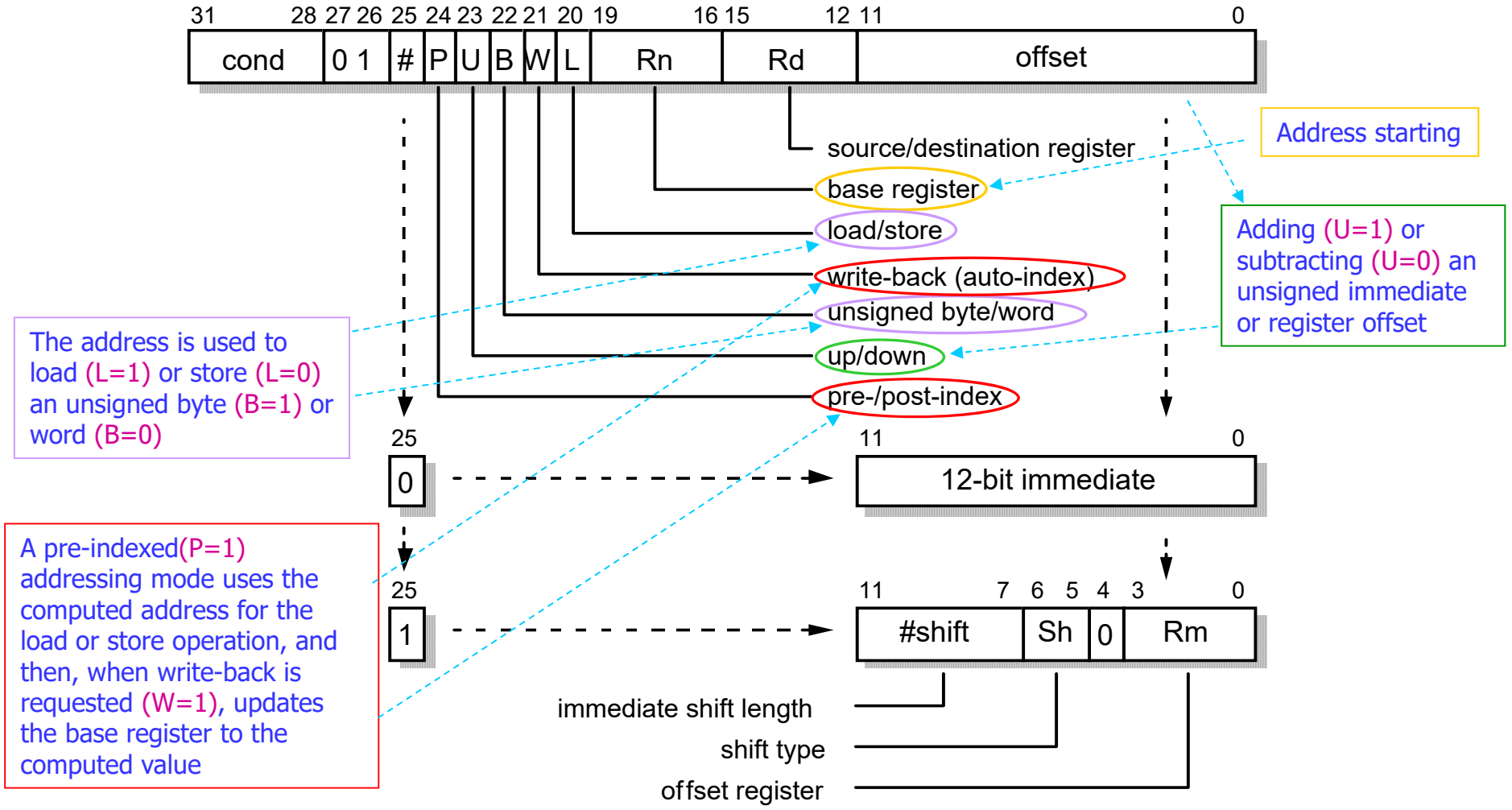
$$immediate = (0 \rightarrow 255) \times 2^{2n}, \quad \text{where } 0 \le n \le 12$$

4.3 ARM instruction execution

Data transfer instructions

- A data transfer (load or store) instruction computes a memory address in a manner very similar to the way a data processing instruction computes its result
- A register is used as the base address, to which is added (or from which is subtracted) an offset which again may be another register or an immediate value (12-bit)
- The address is sent to the address register, and in a second cycle the data transfer takes place (Fig. 4.3)
- Rather than leave the datapath largely idle during the data transfer cycle, the ALU holds the address components from the first cycle and is available to compute an auto-indexing modification to the base register if this is required

```
LDR   r0,  [ r1 , #4 ] !      ; r0 := mem32 [ r1 + 4 ]
                               ; r1 := r1 + 4
```

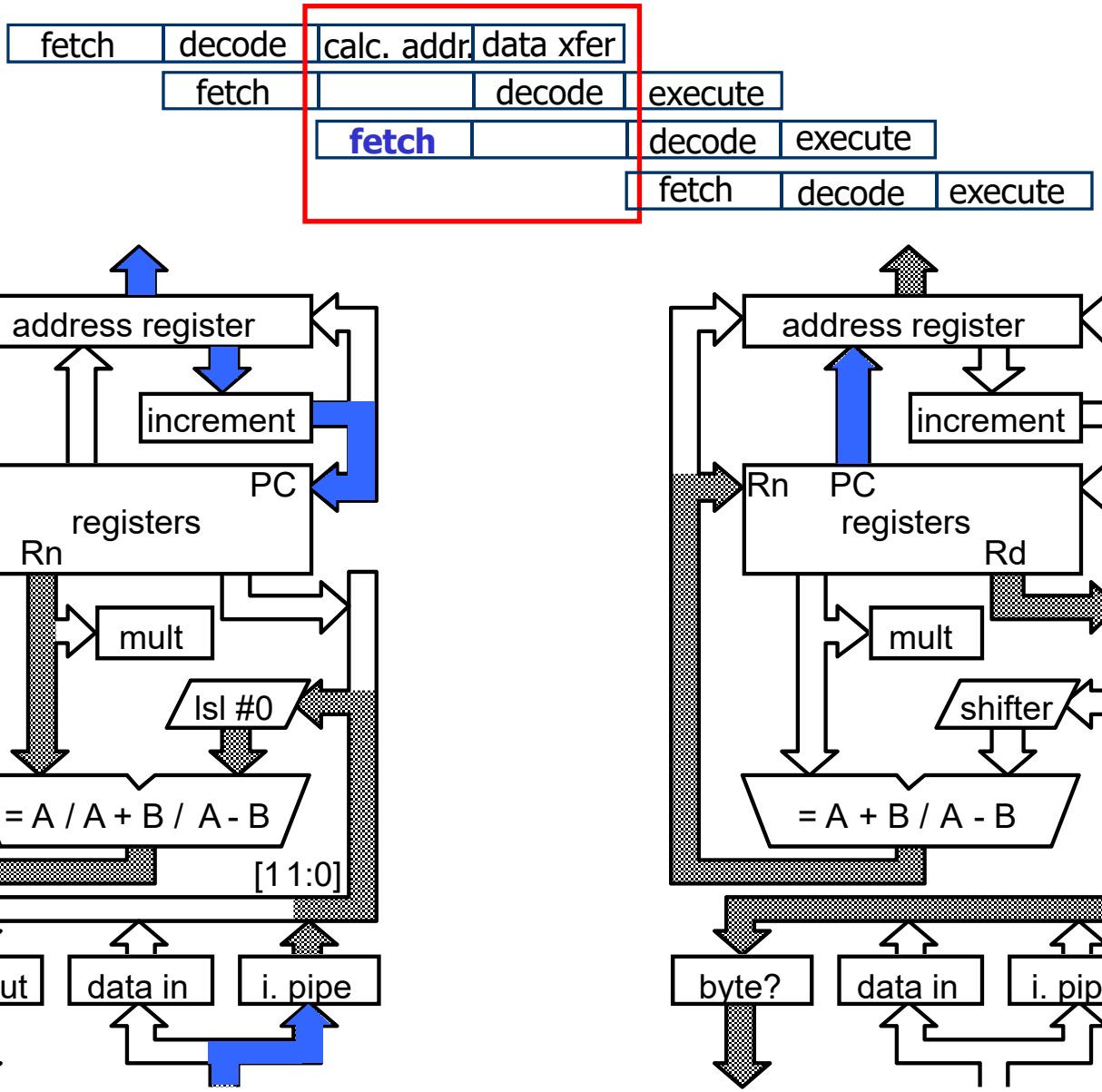


LDRLS pc, [r1, r0, LSL #2] Chap.3, p.35

Fig 5.9 Single word and unsigned byte data transfer instruction binary encoding

4.3 ARM instruction execution

- Data transfer instructions (Cont.)
  - The datapath operation for the two cycles of a data store instruction (STR) with an immediate offset are shown in **Fig. 4.6**
  - The incremented PC value is stored in the register bank at the end of the first cycle so that the address register is free to accept the data transfer address for the second cycle
  - At the end of second cycle the PC is fed back to the address register to allow instruction prefetching to continue
  - The value sent to the address register in a cycle is the value used for the memory access in the following cycle
  - The address register is a pipeline register between the processor datapath and the external memory



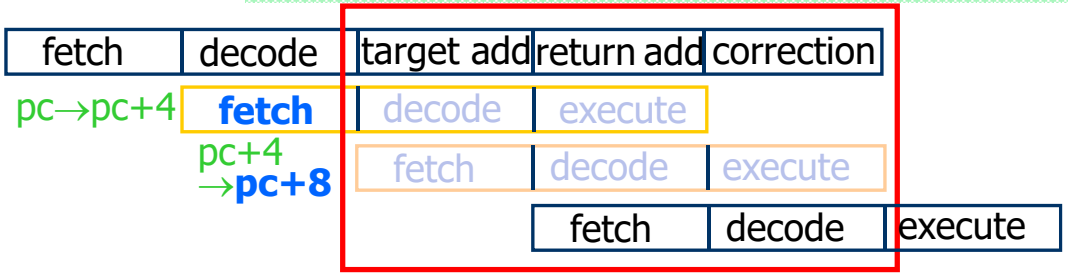
(a) 1st cycle - compute address *(b) 2nd cycle - store data & auto-index*

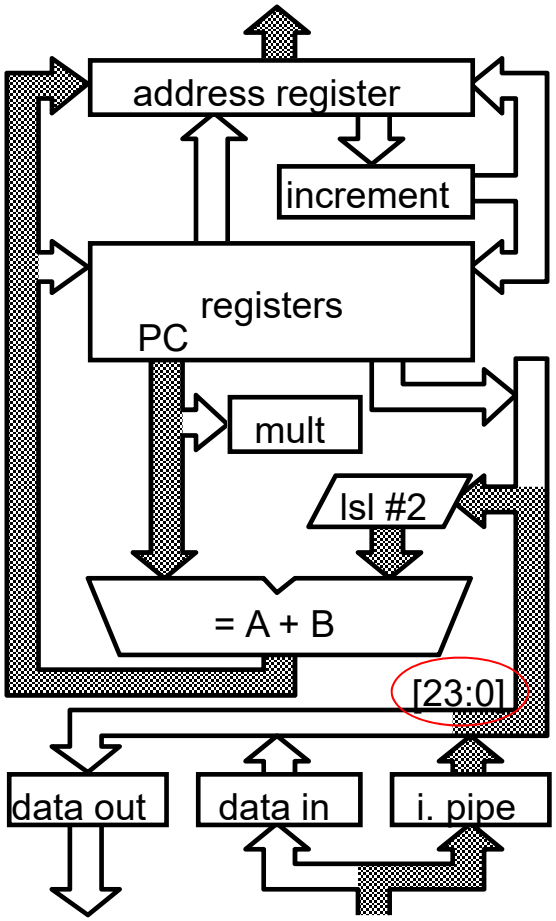
Fig. 4.6 STR (store register) datapath activity

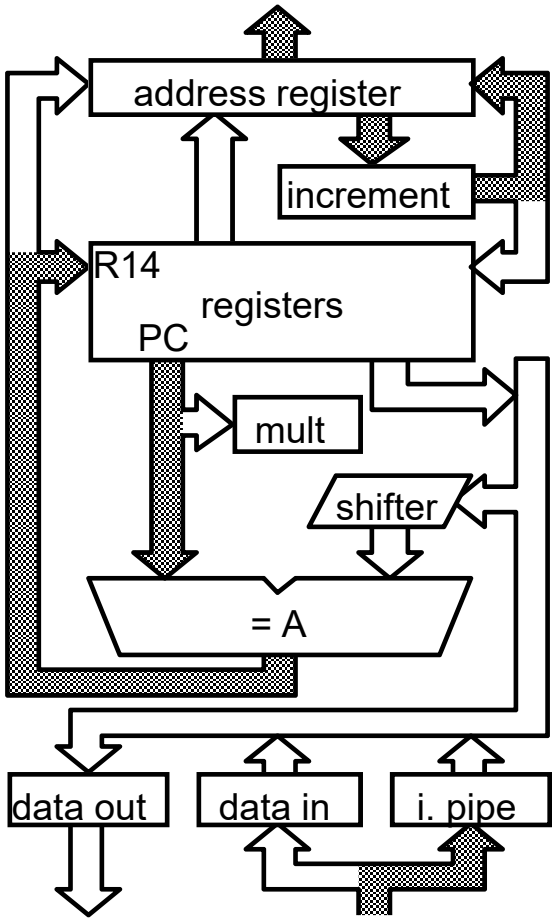
4.3 ARM instruction execution

Branch instructions

- Compute the target address in the first cycle as shown in Fig. 4.7
  - A 24-bit immediate field is extracted from the instruction and then shifted left two bit positions to give a word-aligned offset which is added to the PC
- The second cycle: the result is issued as an instruction fetch address, and while the instruction pipeline refills the return address is copied into the link register (r14) if this is required (instruction is a 'branch with link')
- The third cycle, which is required to complete the pipeline refilling, is also used to make a small correction to the value stored in the link register in order that it points directly at the instruction which follows the branch
  - This is necessary because r15 contains pc+8 whereas the address of the next instruction is pc+4







(a) 1st cycle - compute branch target *(b) 2nd cycle - save return address*

Fig. 4.7 The first two (of three) cycles of a branch instruction